Attorney Docket No. 42390.P13869 PATENT

UNITED STATES PATENT APPLICATION

FOR

METHOD FOR REBALANCING RESOURCES WITHIN A GLOBAL RESOURCE NAMESPACE

Applicants:

David A. Wyatt

Prepared by:

Calvin E. Wells
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN
2200 Mission College Boulevard
Santa Clara, CA 95052

METHOD FOR REBALANCING RESOURCES WITHIN A GLOBAL RESOURCE NAMESPACE

Field Of The Invention

[0001] The present invention pertains to the field of computer systems. More particularly, this invention pertains to the field of managing physical and virtual resources within multi-function, integrated devices.

Background of the Invention

[0002] Many of today's computer systems utilize highly-integrated chipsets that include a multiplicity of functional units that are shared among a variety of software clients such as device drivers. These software clients may also place demands on other system resources such as memory bandwidth. Problems may occur in the system if too many demands are placed on a system resource. For example, for computer systems that use part of the main system memory to store graphics or video data, if the various device drivers place a demand on the main system memory that exceeds the main system memory bandwidth capabilities, then visible artifacts may occur on a display screen as a result of excessive latencies experienced by a graphics controller trying to access main system memory.

[0003] Computer system designers try to solve these problems by anticipating the needs of the various system devices and providing adequate resources. Additional bandwidth resources can be obtained by increasing clock speeds, widening interfaces, improving communication protocols, etc. Device driver designers try to understand the available resources and design the software accordingly. These approaches are not without their own set of difficulties. Computer system designers cannot anticipate all of the ways that a system may be used. Device driver designers must tailor their products with a particular system configuration in mind in order to take advantage of

available resources. This forces device driver designers to revise their products for every new chipset or system configuration.

[0004] Another problem results from the fact that a device driver has no knowledge of what other device driver components or other device drivers are doing, or what system resources are being used by other device driver execution threads. If a particular system resource is already being heavily used and a device driver requires additional use of that resource, the device driver will go ahead and try to use the resource even if it means that the demands on that resource will exceed the resource's capabilities (such as with the main system memory bandwidth example mentioned above). Further, with the type of system described above, there are no provisions for reacting to changing computer system demands and restraints.

[0005] Accordingly, it would be desirable to have a means for managing, predicting, and reacting to computer system demands and restraints whereby device drivers would have access to enough computer system functionality information to ensure that device driver demands remain with the limitations of the hardware.

Brief Description of the Drawings

[0006] The invention will be understood more fully from the detailed description given below and from the accompanying drawings of embodiments of the invention which, however, should not be taken to limit the invention to the specific embodiments described, but are for explanation and understanding only.

[0007] Figure 1 is a block diagram of one embodiment of a computer system that includes a highly integrated system logic device.

[0008] Figure 2 is a diagram of an example namespace for a system logic device including a graphics controller that uses a portion of main system memory to store graphics data.

[0009] Figure 3 is a diagram of an example namespace for a system logic device including a graphics controller that uses local graphics memory to store graphics data.

[0010] Figure 4 is a diagram of a global resource namespace within a resource manager where the resource manager is set up in a server/client arrangement.

[0011] Figure 5 is a flow diagram of one embodiment of a method for managing physical and virtual resources within a multifunction integrated chipset.

[0012] Figure 6 is a flow diagram of an additional embodiment of a method for managing physical and virtual resources within a multifunction integrated chipset.

[0013] Figures 7a and 7b together form a flow chart of one embodiment of a method for rebalancing resources within a global resource namespace.

[0014] Figures 8a and 8b together form a flow diagram of one embodiment of a method for rebalancing resources within a global resource namespace.

Detailed Description

[0015] In general, the example embodiments discussed below describe a technique for managing physical and virtual resources within a multifunction chipset. The embodiments discussed below include a resource manager in the form of a software agent that maintains a global resource namespace built from a list of parent-child object relationships. In general, the parent objects represent resource producers while the child objects represent resource consumers. Examples of physical resources include functional units such as graphics controller rendering engines, digital video output units, digital display outputs, video capture ports, etc. An example of a virtual resource is memory bandwidth. By examining the various parent-child relationships and their associated physical and virtual resources in the global resource namespace, the resource manager can determine how the various system resources are being consumed and balance the net available parent resources globally, as well as across the individual child consumers. Interfaces are provided whereby software drivers and driver components can gain access to the global resource namespace information through the resource manager.

[0016] The resource manager may be implemented according to a server-client model. The client portion of the resource manager can be included as part of the system's various device drivers. The clients can make calls to the server portion of the resource manager to perform various tasks involving resource management. Some of these tasks are described below. For the example embodiments described herein, there is only one instance of the resource manager server and only one instance of the global resource namespace while there are multiple instances of the resource manager clients.

[0017] An additional general explanation of the embodiments is as follows. The embodiments break the system down into classes of producers and consumers. The production of bandwidth is determined by the initial static configuration of the chipset. As consumers (child objects) are attached to producers (parent objects), the resource manager maintains a record of consumed resources as well as remaining or available resources. Feature assignments can then be treated as allocation requests and recorded in a global table. The embodiments compute the bandwidth and demands of each feature and subsystem allocations in real-time, tracking changes in the production as it is affected by external events (such as clock throttling or clock/voltage scaling). The embodiments determine the requirements of potential resource allocations, compare it to the record of net available resources, and provide simple answers to the question of whether a feature, or combination of features, is possible within the given system constraints. The embodiments also take feedback from system and user events as well as policy and uses this information to modify the state model of the system.

[0018] Among the intended benefits of the resource manager as described herein is that the various software device drivers do not need to be specially coded with any particular system configuration in mind since the abstracted system configuration information can be determined using the resource manager. Device driver designers can maximize the reuse of code and reduce development time and support efforts because there is less of a need to specifically tailor the drivers for any particular system configuration. The maximum reuse of code can also reduce validation efforts and improve software stability.

[0019] Another intended benefit of the resource manager is the ability to maximize resource usage without placing demands on a resource that exceed that

resource's capabilities. For example, in a computer system there may be several system agents requiring access to main memory. The resource manager keeps a record of the amount of available memory bandwidth and also keeps a record of how much of that bandwidth is being consumed by the several resources requiring access to main memory. When an additional system agent requires access to main memory, the agent's software device driver inquires of the resource manager to discover whether enough main memory bandwidth remains to be able to support the agent's requirements. If there is not enough bandwidth remaining, the device driver can make intelligent decisions on how best to proceed.

[0020] Although the embodiments discussed herein describe a resource manager implemented in software, other embodiments are possible using hardware implementations. Further, although the embodiments discussed below mention the management of resources included in and surrounding a graphics controller included as part of a highly-integrated multifunction chipset, other embodiments are possible that manage other types of computer system resources.

[0021] Figure 1 is a block diagram of one embodiment of a computer system that includes a highly integrated system logic device 120. System logic devices of this type are often referred to as "chipsets." The system logic device 120 is couple to a processor 110. The system logic device 120 includes a memory controller 122 and a graphics controller 124. The memory controller 122 is coupled to a main system memory 130. The memory controller 122 is also coupled to the graphics controller 124 and is further coupled to an optional advanced graphics port (AGP) graphics device 180. The AGP graphics device 180 can either complement or replace the graphics controller 124.

[0022] The graphics controller 124 may have many functional units including, but not limited to, rendering engines, blitter engines, video capture port units, digital display output units, digital video output units, CRT display output units, overlay units, cursor units, plane units, encoding/decoding units, etc. The graphics controller 124 may include more than one of each of these units. For purposes of example, the graphics controller 124 is shown in Figure 1 as being coupled to a CRT display 140 and a digital display device 150.

[0023] The graphics controller 124 is also optionally coupled to a graphics local memory 190. The graphics local memory 190 is used to store graphics data. If the graphics local memory 190 is not installed, then the graphics controller will use a portion of the system memory 130 to store graphics data. Embodiments are also possible where graphics data is stored both in the graphics local memory 190 and the system memory 130.

[0024] Also coupled to the system logic device 120 is an input/output hub 160. The input/output hub 160 is further coupled via a peripheral component interconnect (PCI) bus 165 to a PCI device 170. It is possible for more than one device to be attached to the PCI bus 165. The types of devices that may be attached to the PCI bus include, but are not limited to, disk drives or other storage devices. These devices typically include data that needs to be moved to or from system memory 130.

[0025] The system of Figure 1 includes several functional units or devices that require access to system memory 130. For example, the processor 110 requires access to system memory, as does the graphics controller 124 as well as devices coupled to the input/output hub 160. The resource manager as described herein can be used to manage the consumption of the system memory bandwidth.

[0026] Figure 2 is a diagram of an example global resource namespace for the system logic device 120 where the graphics controller 124 uses a portion of system memory 130 to store graphics data (the optional graphics local memory 190 is not installed). A system logic device object 202 represents the root of the namespace tree structure. A system memory object 204 is linked to the system logic device object 202. The system memory object 204 may include information to link the system memory object 204 to the system logic device object 202, as well as information uniquely identifying the system memory object 204. Information regarding system memory bandwidth may also be included as part of the system memory object 204.

[0027] Linked to the system memory object 204 are a display pipe object 206 and a capture port object 208. The display pipe object 206 represents a particular display pipe within the graphics controller 124. The graphics controller 124 may include more than one display pipe. The display pipe object 206 includes information linking the object to the system memory object 204. The display pipe object 206 also includes information that uniquely identifies the object and further may include information regarding bandwidth consumption or information regarding the rate at which pixel data is clocked through the display pipe. The display pipe object 206 may further include other information that describes various other features, capabilities, or requirements of the display pipe.

[0028] The capture port object 208 represents a video capture functional unit within the graphics controller 124. The capture port object 208 includes information linking the object to the system memory object 204. The capture port object 208 also includes information that uniquely identifies the object and further may include information regarding bandwidth consumption or information regarding the rate at which video data is transferred through the capture port. The capture port object 208

may further include other information that describes various other features, capabilities, or requirements of the capture port.

[0029] Linked to the display pipe object 206 are a digital video output (DVO) object 210, a plane object 212, an overlay object 214, and a cursor object 216. The DVO object 210 represents a digital video output unit within the graphics controller 124. The plane object 212 represents a display plane unit within the graphics controller 124. The overlay object 214 represents an overlay unit within the graphics controller 124. The cursor object 216 represents a hardware cursor unit within the graphics controller 124. The objects 210, 212, 214, and 216 include information that links the objects to the display pipe object 206. The objects 210, 212, 214, and 216 also include information regarding the bandwidth consumption properties of the respective functional units. The objects 210, 212, 214, and 216 may further include other information that describes various other features, capabilities, or requirements of the respective functional units.

[0030] Linked to the capture port object 208 is a capture coder/decoder (codec) object 218 that represents a video capture codec unit within the graphics controller 124. The capture codec object 218 includes information linking the capture codec object 218 to the capture port object 208. The capture codec object 218 also includes information regarding the bandwidth consumption properties of the video capture codec unit. The capture codec object 218 may also include information that describes other features, capabilities, or requirements of the video capture codec unit.

[0031] The resource manager can determine the net available bandwidth of the system by walking the various branches of the tree structure shown in Figure 2 and observing the bandwidth production or consumption information included in the various objects.

[0032] Figure 3 is a diagram of an example namespace for the system logic device 120 including the graphics controller 124 where the graphics controller 124 uses the local graphics memory 190 to store graphics data. A system logic device object 302 represents the root of the namespace tree structure. A system memory object 320 is linked to the system logic device object 302. Although in the example embodiment of Figure 3 no objects are shown attached to the system memory object 320, other embodiments are possible where objects representing functional units within the system logic device 120 are attached to the system memory object 320.

[0033] A local graphics memory object 304 is linked to the system logic device object 302. The local graphics memory object 304 may include information to link the local graphics memory object 304 to the system logic device object 302, as well as information uniquely identifying the local graphics memory object 304. Information regarding local graphics memory bandwidth may also be included as part of the local graphics memory object 304.

[0034] Linked to the local graphics memory object 304 are a display pipe object 306 and a display pipe object 308. The display pipe objects 306 and 308 represent two display pipes within the graphics controller 124. The display pipe objects 306 and 308 include information linking the objects to the local graphics memory object 304. The display pipe objects 306 and 308 also include information that uniquely identifies the objects and further may include information regarding bandwidth consumption or information regarding the rate at which pixel data is clocked through the display pipes. The display pipe objects 306 and 308 may further include other information that describes various other features, capabilities, or requirements of the display pipes.

[0035] Linked to the display pipe object 306 are a digital-to-analog converter (DAC) object 310, a plane object 312, an overlay object 314, and a cursor object 316. The DAC object 310 represents a digital-to-analog converter unit within the graphics controller 124. The plane object 312 represents a display plane unit within the graphics controller 124. The overlay object 314 represents an overlay unit within the graphics controller 124. The cursor object 316 represents a hardware cursor unit within the graphics controller 124. The objects 310, 312, 314, and 316 include information that links the objects to the display pipe object 306. The objects 310, 312, 314, and 316 also include information regarding the bandwidth consumption properties of the respective functional units. The objects 310, 312, 314, and 316 may further include other information that describes various other features, capabilities, or requirements of the respective functional units.

[0036] Linked to the display pipe object 308 is a DVO object 318 that represents a digital video output unit within the graphics controller 124. The DVO object 318 includes information linking the DVO object 318 to the display pipe object 308. The DVO object 318 also includes information regarding the bandwidth consumption properties of the digital video output unit. The DVO object 318 may also include information that describes other features, capabilities, or requirements of the digital video output unit.

[0037] The resource manager can determine the net available local graphics memory bandwidth by walking the various branches of the tree structure that are attached to the local graphics memory object 304 and observing the bandwidth production or consumption information included in the various objects of those branches.

[0038] Although the example namespaces of Figures 2 and 3 describe the namespaces as linked-lists, other embodiments may use other techniques to store and track data.

[0039] Figure 4 is a diagram of a global resource namespace 452 within a resource manager where the resource manager is set up in a server/client arrangement including a server portion 450 and client portions 432, 422 and 442. For this example embodiment, the resource manager is implemented in software. The client portions 432, 422, and 442 of the resource manger are compiled and linked into device drivers 430, 420, and 440. A computer system may include a wide variety of software device drivers that manage the operation of a wide variety of system devices and/or functional units. For this example, device driver 420 is a display driver that manages the function of a graphics controller such as the graphics controller 124 of Figure 1. The device drivers 430 and 440 may be device drivers for any of a wide variety of system devices or functional units.

[0040] The display driver 420 may receive input from a user interface 410. The user interface allows a computer system operator to specify display parameters such as display resolution and screen refresh rate.

[0041] As mentioned above, the resource manager can follow a client/server model. Multiple types of software drivers, and additionally multiple instances of similar drivers (e.g., one instance of a display driver per display output) can be clients. In these example embodiments, there is only a single server interface, as represented by the resource manager server 450 in Figure 4. A variety of communication channels may be used to allow the clients (such as clients 432, 422, and 442) to access the resource manager server 450. The communication channels may include driver escapes, I/O control packets, and direct call dispatches. The clients 432, 422, and 442

communicate with a single instance of the resource manager server 450. The resource manager server 450 operates on a single global resource namespace 452. The global resource namespace 452 is initialized in this embodiment with system logic device and platform specific parameters, which may be hard-coded depending on device/revision identifiers, and clarified with information set by fuses within on-chip capability registers. Additionally, parameters typically stored in non-volatile memory such as within the system basic input/output system (BIOS) firmware may be used to further fine-tune the fundamental chip parameters with platform specifics.

[0042] The clients 432, 422, and 442 may request the resource manager server 450 to perform a number of resource management routines. Some of these routines are described as follows.

[0043] One group of routines involves querying the server interface and adding or deleting a resource reference. These routines facilitate establishing a connection between the clients 432, 422, 442, and the server 450. Upon adding a resource reference (registration), each client is tracked and it's resource allocations are tagged. Upon deleting a reference (disconnection), any resources or allocations associated with the client are purged.

[0044] Another group of routines allows real-time events within the system to be reflected in the global resource namespace. For example, system events that cause clock throttling or voltage or clock scaling may be signaled to the resource manager 450 and the resource manager 450 can make adjustments to the global resource namespace 452 to reflect the changes to affected resources.

[0045] Other software routines may allow the clients to query the resource manager server for various purposes. For example, queries may be provided to: allow a client to test if a given resource is available; allow the client to test if a given

resource is attached to a given parent; check if a resource can be attached to a parent; attach a child resource to a parent using parameters provided by a client; freeing an attachment, releasing the child and its consumed resources; allow a client to pre-allocate (reserve) a child/resource, and the bandwidth necessary, as specified by given parameters, without actually completing the attachment of the resource; reverse a previous pre-allocation (reservation); and alter the parameters of an existing resource attachment, allowing it's resource consumption to be increased or decreased.

[0046] Other possible software routines allow for resource manager maintenance. One routine assesses the resource consumption of a node, parent, or branch. Another routine enumerates all of the child resources and parameters attached to the given parent or parents. This routine creates a linked-list of resources which are returned together to the calling client. An additional routine allows a client to test alternative system configurations.

[0047] Figure 5 is a flow diagram of one embodiment of a method for managing physical and virtual resources within a multifunction integrated chipset. At block 510, a record of available resources is maintained. A record of consumed resources is maintained at block 520. Relationships among producers and consumers are tracked at step 530. Finally, at block 540, the record of available and consumed resources is updated upon a change in relationship among producers and consumers.

[0048] Figure 6 is a flow diagram of an additional embodiment of a method for managing physical and virtual resources within a multifunction integrated chipset. At block 610, a list of physical resource objects is stored. A list of virtual resource objects is stored at block 620. A list of parent and child objects is stored at block 630. Finally, at block 640, a tree of relationships for the parent objects and the child objects is created.

[0049] Figures 7a and 7b together form a flow diagram of an embodiment of a method for attaching a resource to a parent within a global resource namespace. The attachment process begins at block 702 with a resource request. This request comes in the form of a call from a software client. The client typically wishes to attach a device or a functional unit to a system resource. If successful, the newly attached client device or functional unit will be represented as a child object within the global resource namespace. The child object will be attached to a parent resource object that represents the system resource.

[0050] At block 704 the global resource namespace is checked and at 706 a determination is made as to whether the requested resource exists in the namespace. If the resource is found to not exist within the namespace, the attach resource software routine fails and the client is notified of the failure. If, however, the resource is found to exist, then at block 708 a determination is made as to whether the resource is available. If the resource is not available, then the procedure fails. If the resource is available, then at block 710 a check is made to determine if any conflicts exist that would prevent the child object from being attached to the parent resource. If a conflict is found, then the process fails. If no conflicts are found, then at block 712 the resource requirements of the child object are calculated. A comparison is then made between the requirements and the available bandwidth, which is calculated at block 714. At block 716, a determination is made as to whether sufficient bandwidth exists to support the child object's requirements. If sufficient bandwidth in not found, then the process fails and the client is notified. If sufficient bandwidth is found, then at block 718 the child resource is attached to the parent resource in the global resource namespace. The child object's bandwidth consumption properties are then feed back to block 714 so that future available bandwidth calculations can consider the

bandwidth consumption properties of the newly-attached child resource. Following the attachment at block 718, the client is notified of the successful procedure at block 720.

[0051] Figures 8a and 8b together form a flow diagram of one embodiment of a method for rebalancing resources within a global resource namespace. The rebalancing process re-computes the resource consumption information including the demands of all parent-child attachments. The process also frees any dangling resources left by clients which have been previously removed. The rebalancing process can be triggered by events in the system that alter the resource consumption properties of the system. Such events may include a resume from suspend mode, a hot-plug event, a display switch, or a docking event. Other types of events that may prompt the rebalance process include changes in power or performance state and thermal or throttling events. Still other types of system events may prompt the rebalancing process.

[0052] The rebalance process begins at block 802. At block 804, available resources are recalculated. At block 806, a determination is made as to whether there is less available resources (after the event that triggered the rebalance process) than is currently consumed. The currently consumed resources can be determined by examining the resource production and consumption information for the various resources listed in the global resource namespace. If the available resources are not less than the resources currently consumed, then the global settings are adjusted at block 808 and the process ends successfully. In this case, there was no need to rebalance the resources within the global resource namespace.

[0053] If, however, there are less available resources than are being currently consumed, then at block 810 a temporary namespace is allocated. Beginning at block

812, each child resource will be examined. The child enumeration occurs at block 812. Block 814 indicates that if there are no more resources (all of the child objects have been examined), then at block 816 the old namespace is destroyed and at block 818 the temporary namespace becomes the new namespace and the process completes successfully.

[0054] Block 814 also indicates that if there are additional resources that require examination, then processing proceeds to block 820. At block 820, a determination is made as to whether an owner exists for the current child resource. If no owner exists, then processing returns to block 812. If, however, an owner does exist for the current child resource, then the resource is attached to the owner in the temporary namespace at block 822. The attachment process indicated by block 822 may be implemented as disclosed above in connection with Figures 7a and 7b. Block 824 indicates that if the attachment of block 822 was successful, then processing returns to block 812. If the attachment process of block 822 was not successful, then processing proceeds to block 826. At block 826 a determination is made as to whether the owner of the current child resource can resubmit an attachment request. If the owner can resubmit and if the owner (client) supports notification, then at block 828 the client is notified to resubmit.

[0055] Processing then returns to block 812. If the owner cannot resubmit, then at block 830 the temporary namespace is destroyed. At block 832 the temporary allocations are freed. The process then ends with an unsuccessful result.

[0056] It is anticipated that there may be clients of the global resource namespace which compete for resources. These clients may asynchronously and concurrently create and remove parent-child & producer-consumer relationships. For example, clients which submit a request which fails at one instance due to limited

availability may succeed later if another client has released more resources or when a rebalance procedure has occurred. One way to detect this is for clients to repeatedly submit resource requests which may eventually succeed. However, this process adds unnecessary calling and computation overhead if nothing has changed. To help solve this problem, the resource manager may provide to clients a token value as a unique identifier representing the entire current configuration. The token may be referred to as the Configuration-Unique Identifier (CUID). This token will be unique for the current parent-child resource configuration, and will vary the moment any resource allocation or release occurs. The resource manager may provide this value immediately to any client, thereby allowing comparison of any two successive identifier values. This comparison enables an immediate determination of whether any resource configurations have changed or of whether nothing has changed, without necessitating re-evaluating the entire tree of resources. This identifier may be created from a random number generator or other source with low chance of collision. It may also be created from a one-way hash of the configuration space, since repeatability for identical configurations is a desirable feature.

[0057] The available resources are affected by external, as well as internal factors. Without any changes occurring as a result of client resource allocation or release, the balance of the system may change. For example, whereas a client allocation may have initially failed, a change in system bandwidth due to a change in core clock speed, or thermal throttles may allow a subsequent allocation to succeed. This issue may be addressed without the overhead of clients repeatedly submitting and failing requests. Upon initialization, and for any configuration change, the resource manager may generate a token value being a unique identifier, with a very low probability of collision, based on the current system parameters. This token may be

referred to as the System-Unique Identifier (SUID). The resource manager may provide this value immediately on request, allowing client comparison of any two successive values to quickly determine if any changes have occurred. While the CUID & SUID may be used separately for different purposes, together the two identifiers can effectively summarize the state of the system, resources, and the global resource namespace, and can therefore allow clients to quickly determine if an assumption remains valid from one point in time to the another. For example, a user interface client of the resource manager may enumerate the current configuration and then offer a user selection based on current available resources. Some time later, the same user interface client may again need to offer a user selection. By checking the CUID & SUID, the user interface client can then quickly decide if it can re-display the last selections without concern for the effects of other parallel client activities or hidden resource rebalance.

[0058] In the foregoing specification the invention has been described with reference to specific exemplary embodiments thereof. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the invention as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative rather than in a restrictive sense.

[0059] Reference in the specification to "an embodiment," "one embodiment," "some embodiments," or "other embodiments" means that a particular feature, structure, or characteristic described in connection with the embodiments is included in at least some embodiments, but not necessarily all embodiments, of the invention. The various appearances of "an embodiment," "one embodiment," or "some embodiments" are not necessarily all referring to the same embodiments.